BAB III

ANALISIS DAN PERANCANGAN SISTEM

3.1 Algoritma *Breadth First Search*

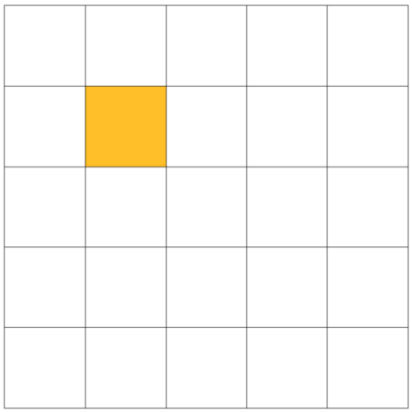
Berikut ini adalah proses yang dilakukan dengan menggunakan algoritma *Breadth first search* untuk pencarian jalur. Proses pencarian ini berdasarkan algoritma *Breadth first search* (Desphande, 2008: 1-23) yaitu

```
unmark all vertices
choose some starting vertex x
mark x
list L = x
tree T = x
while L nonempty
choose some vertex v from front of list
visit v
for each unmarked neighbor w
        mark w
        add it to end of list
                add edge vw to T
```

3.1.1 Proses Pencarian

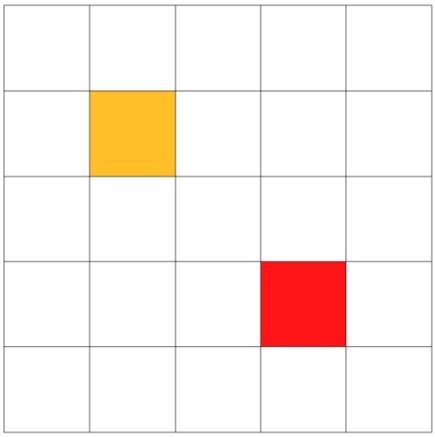
Proses yang diterapkan pada pencarian menggunakan algoritma *Breadth first search* yaitu

1. Menentukan simpul awal.



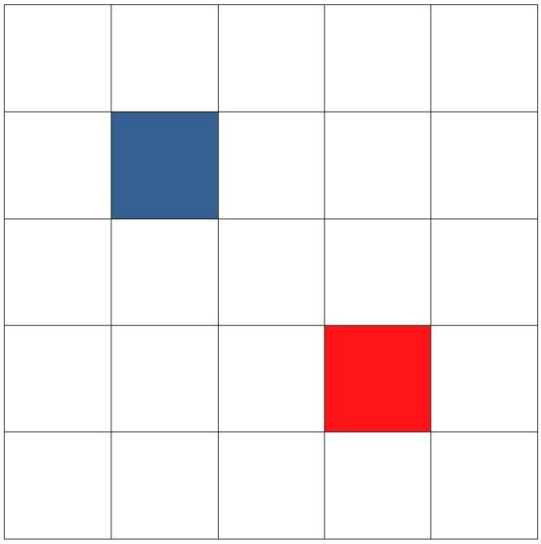
Gambar 3.1– Menentukan simpul awal algoritma *Breadth First Search*

2. Menentukan simpul akhir.



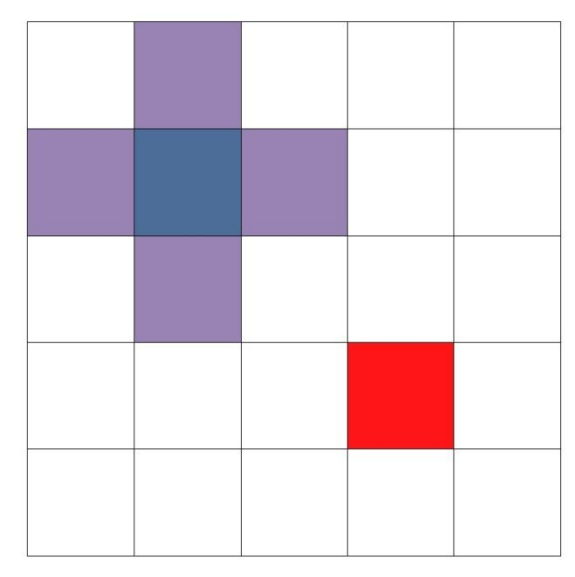
Gambar 3.2 – Menentukan simpul akhir algoritma *Breadth First Search*

3. Jadikan simpul posisi awal sebagai *parent* (simpul yang bewarna biru) dan beri tanda jika telah dikunjungi.



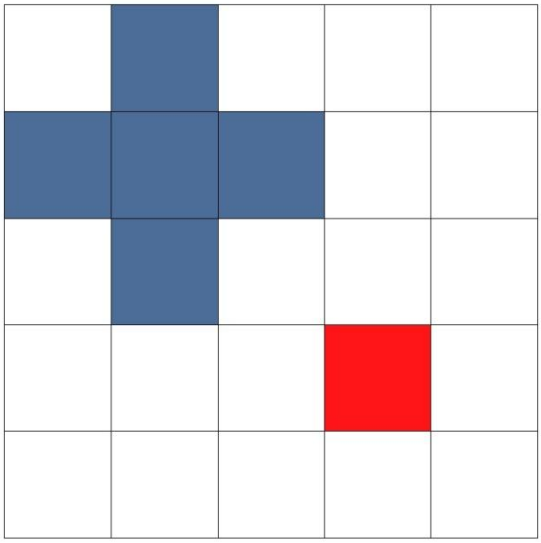
Gambar 3.3 - Jadikan posisi awal sebagai *parent* pada algortima *Breadth First Search*

4. Membuka simpul-simpul lainnya untuk simpul yang telah ditandai sebagai *parent*, jadikan status yang telah terbuka tersebut sebagai *child* (simpul yang berwarna ungu).



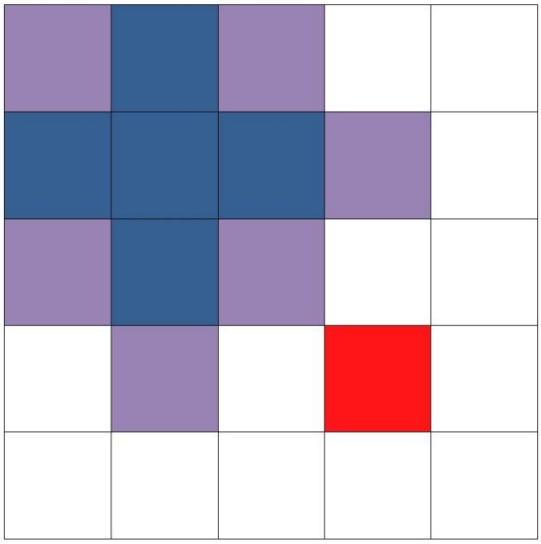
Gambar 3.4 - Membuka simpul yang telah dijadikan status parent pada algoritma *Breadth First Search*

5. Ubah status pada *path* yang terbuka tersebut menjadi telah dikunjungi, dan jadikan statusnya sebagai *parent.*



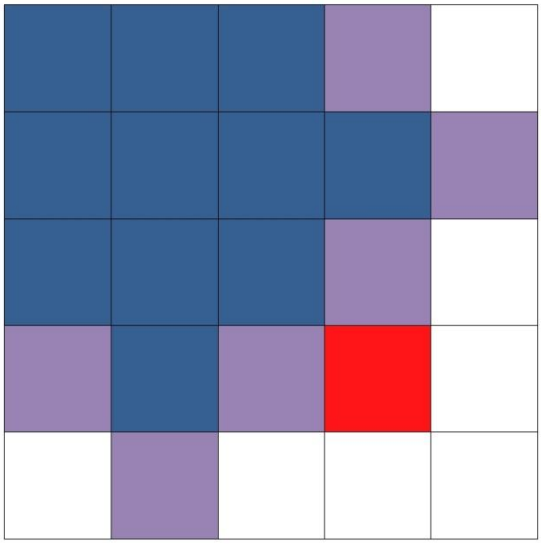
Gambar 3.5 – Merubah status simpul *child* menjadi simpul *parent*

6. Buka path yang statusnya sebagai *parent*, dan *path* yang terbuka tersebut dijadikan statusnya sebagai *child*.

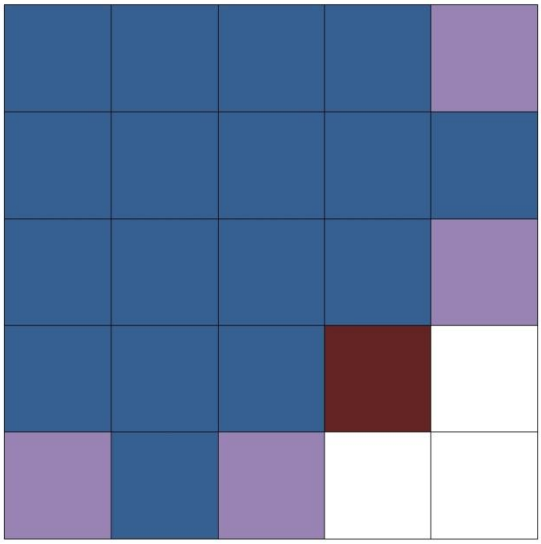


Gambar 3.6 - Membuka simpul yang statusnya telah dijadikan *parent*

7. Lakukan mulai dari langkah ke 2 sampai menemukan posisi akhir.



Gambar 3.7 – Melakukan langkah ke 3 kembali sampai menemukan simpul akhir

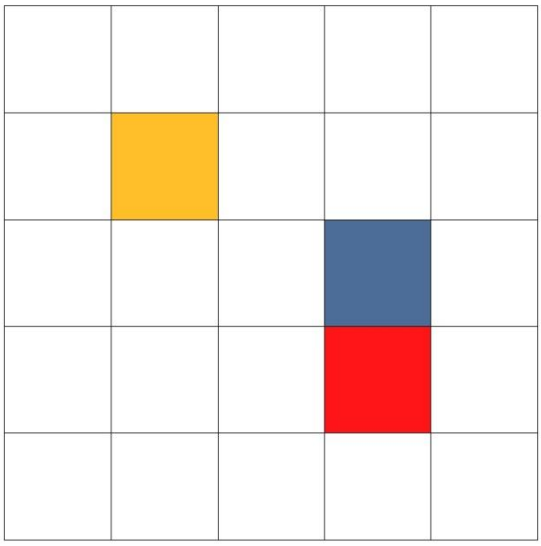


Gambar 3.8 – Setelah menemukan simpul akhir

3.1.2 Menampilkan Hasil Dari Algoritma *Breadth First Search*

Untuk menampilkan hasil dari pencarian *Breadth first search*. Maka dilakukan beberapa proses yaitu

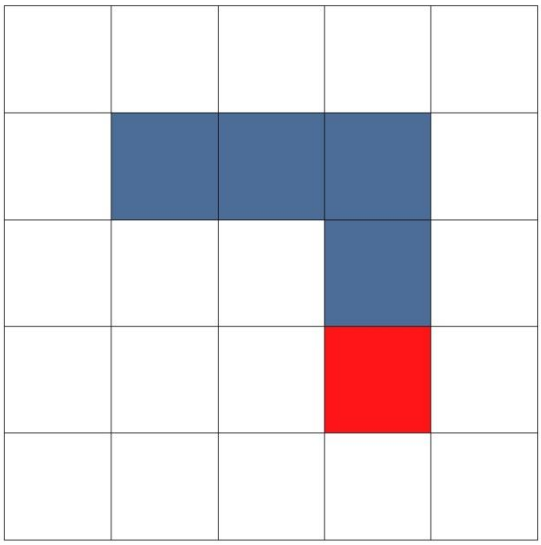
1. Menemukan *parent* dari simpul akhir.



Gambar 3.9 – Menemukan *parent* dari simpul akhir

.

2. Lakukan terus langkah ke 1 sampai ketemu simpul awal.



Gambar 3.10 – Hasil dari pencarian

3.2. Algoritma A*

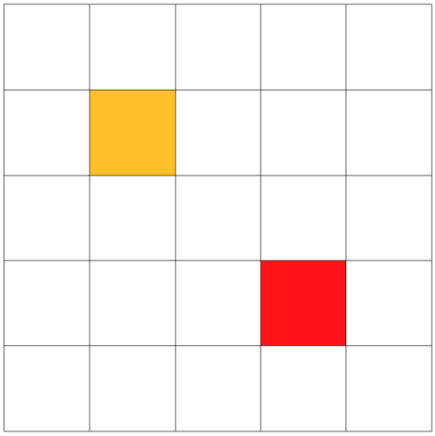
Proses pencarian berdasarkan algoritma A* (Jones, 2008: 57) yaitu

```
Initialize OPEN list (priority queue)
Initialize CLOSED list
Place start node on the OPEN list
Loop while the OPEN list is not empty
    Get best node (parent) from OPEN list (least f (n))
    if parent is the goal node, done
    Place parent on the CLOSED list
    Expand parent to all adjacent nodes (adj_node)
        if adj_node is on the CLOSED list
            discard adj_node and continue
        else if adj_node is on the OPEN list
            if adj_node's g value is better than
                the OPEN.adj_node's g value
                discard OPEN.cur_node
                calculate adj_node's g, h and f values
                set adj_node predecessor to parent
                add adj_node to OPEN list
                continue
            end
        else
            calculate adj_node's g, h and f values
            set adj_node predecessor to parent
            add adj_node to OPEN list
        end
    end
```

end loop

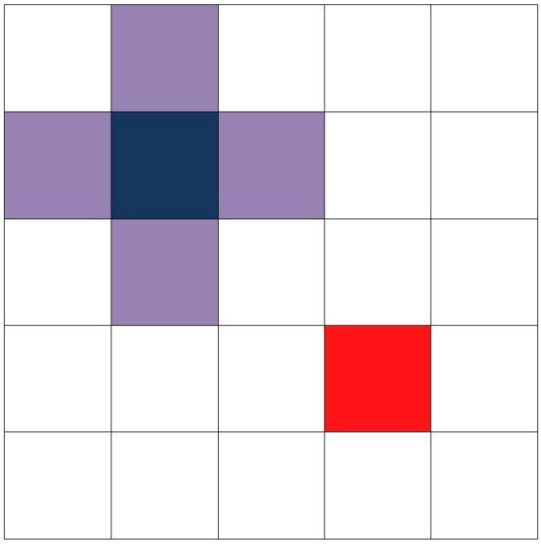
Proses yang dilakukan dalam pencarian menggunakan algoritma A* adalah

1. Menentukan simpul awal dan simpul akhir.



Gambar 3.11 – Awal pencarian A*

2. Tandai simpul tersebut karena telah dikunjungi, membuka simpul lainnya dan menyimpan simpul-simpul tersebut pada *open list.*



Gambar 3.12 – Membuka simpul pada pencarian algoritma A*

3. Menentukan heuristik antar simpul dengan perhitungan *Manhattan Distance* untuk algoritma A*, perhitungan tersebut adalah

   Berdasarkan fungsi heuristik pada algoritma A* maka:

   $$f(n) = g(n) + h(n)$$

   f(n) = fungsi evaluasi (jumlah g(n) dengan h(n)).

   g(n) = biaya (*cost*) yang dikeluarkan dari keadaan awal sampai keadaan n.

   h(n) = estimasi biaya untuk sampai pada suatu tujuan mulai dari n.

   n = *Grid* saat ini.

   D = nilai untuk melakukan pergerakan.

   Nilai h dicari menggunakan fungsi *Manhattan Distance* untuk 4 arah (Rahayu, 2011: 31):

   $$h(n) = D * (abs(n.x-goal.x) + abs(n.y-goal.y))$$

   contohnya misalkan mencari nilai heuristik dari masing-masing simpul yang terbuka pada Gambar 3.11.

dengan simpul awal:

Sx : 1

Sy : 1

dan simpul akhir:

Gx : 3

Gy : 3

Hasil perhitungan:

Menentukan nilai g(n)

Karena dilakukan pertama kali maka nilai g(n) = 1

Menghitung masing-masing nilai heuristik dari *path* yang terbuka

1) Simpul pada (1,0).

h (n) = D * (abs(n.x-*goal*.x) + abs(n.y-*goal*.y))

h (1,0) = 1 * (abs(1-3) + abs(0-3))

= 5

f(1,0) = g(1,0) + h(1,0)

= 1 + 5

=6

2) Simpul pada (0,1).

h (n) = D * (abs(n.x-*goal*.x) + abs(n.y-*goal*.y))

h (0,1) = 1 * (abs(0-3) + abs(1-3))

= 5

f(0,1) = g(0,1) + h(0,1)

= 1 + 5

=6

3) Simpul pada (1,2).

h (n) = D * (abs(n.x-*goal*.x) + abs(n.y-*goal*.y))

h (1,2) = 1 * (abs(1-3) + abs(2-3))

= 3

f(1,2) = g(1,2) + h(1,2)

= 1 + 3

=4

4) Simpul pada (2,1).

h (n) = D * (abs(n.x-*goal*.x) + abs(n.y-*goal*.y))

h (2,1) = 1 * (abs(2-3) + abs(1-3))

= 3

f(2,1) = g(2,1) + h(2,1)

= 1 + 3

= 4

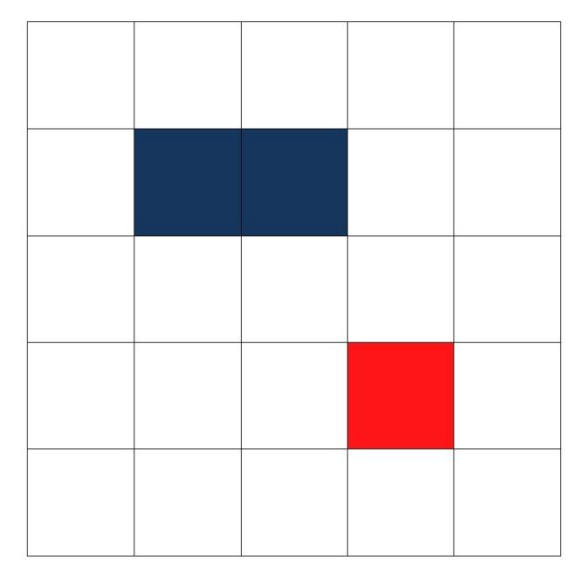
Sehingga nilai heuristik untuk masing-masing simpul adalah

Tabel 2.2 – Hasil perhitungan heuristik A*

Simpul	Nilai Heuristik
1,0	6
0,1	6
1,2	4
2,1	4

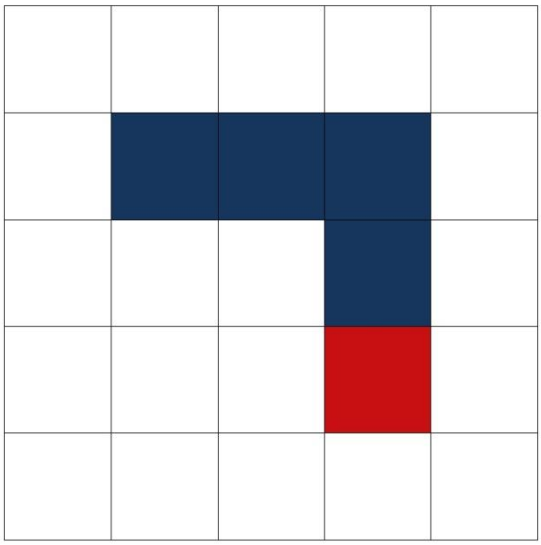
4. Memilih simpul berdasarkan nilai heuristik terkecil.

Karena simpul pada (2,1) dan (2,2) memiliki nilai heuristik, maka boleh memilih salah satu dari simpul-simpul tersebut, berdasarkan pada Gambar 21 misalkan yang terpilih adalah simpul di (2,1).



Gambar 3.13 – Hasil pertama pencarian A*

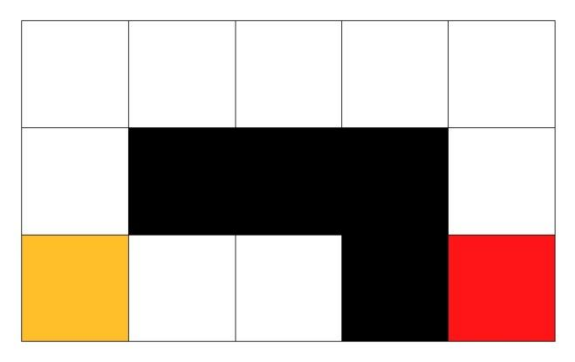
5. Lakukan terus langkah ke 2 sampai menemukan simpul akhir atau *open list* kosong.



Gambar 3.14 – Hasil pencarian algoritma A*

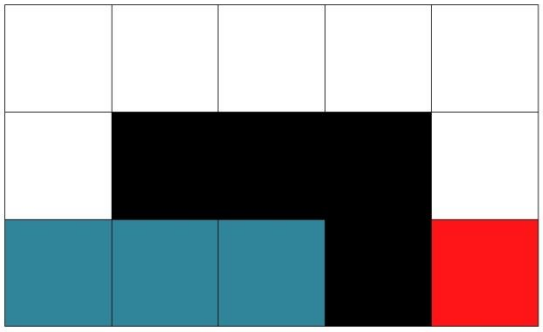
3.2.1 Runut Balik Pada Algoritma A*

Pada *block maze*, sering sekali algoritma A* salah menemukan simpul akhir dikarenakan memilih nilai heuristik terkecil. Misalkan pada kasus mencari jalur dari simpul awal ke simpul akhir dengan terdapat simpul-simpul yang tidak bisa dilalui seperti gambar di bawah ini:



Gambar 3.15 – Awal pencarian algoritma A* yang menggunakan kasus runut balik

Maka dengan menggunakan perhitungan heuristik *Manhattan Distance* solusi tidak dapat ditemukan.

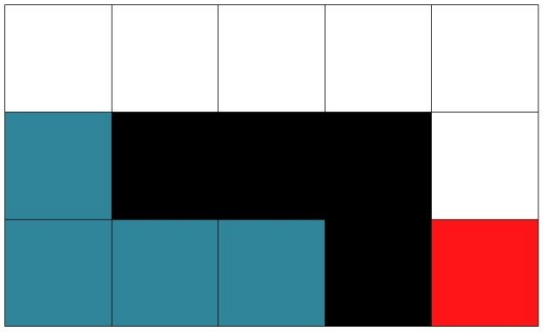


Gambar 3.16 – Kekeliruan pada algoritma A*

Oleh karena itu, dilakukan runut balik, yang dilakukan dengan cara:

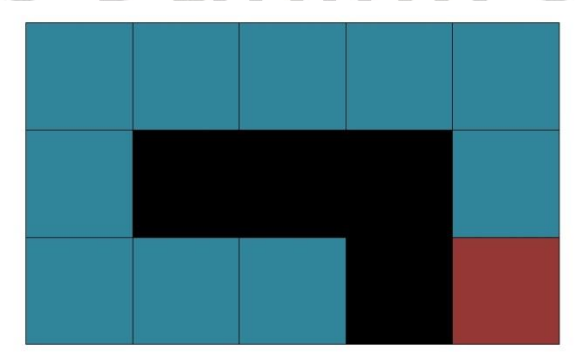
1. Menyimpan setiap simpul yang bercabang pada *stack* tertentu
2. Apabila daftar simpul pada *open list* kosong maka pilih suatu simpul dari *stack* tersebut.
3. Buka simpul-simpul tetangga yang tidak dikunjungi dari simpul yang terpilih.
4. Lakukan pencarian simpul dan penentuan nilai heuristik terkecil seperti pada algoritma A*.
5. Simpan hasil simpul yang dicari dengan nilai heuristik terkecil pada *open list.*

Dengan menggunakan algoritma di atas maka pencarian tersebut menjadi:



Gambar 3.17 – Runut balik pada algoritma A*

Dan karena *open list* telah terisi dari proses runut balik yang telah dilakukan tersebut maka pencarian dilakukan seperti biasanya pada algoritma A*, sehingga hasilnya adalah

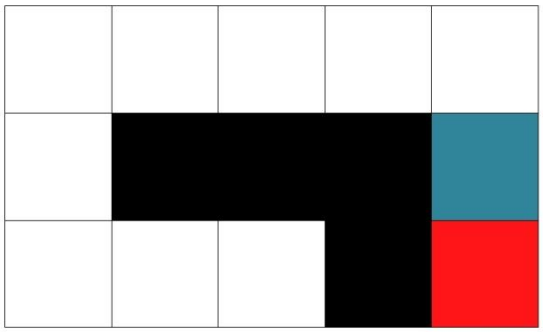


Gambar 3.18 – Hasil pencarian algoritma A*

3.2.2 Menampilkan Hasil Pencarian Menggunakan Algoritma A*

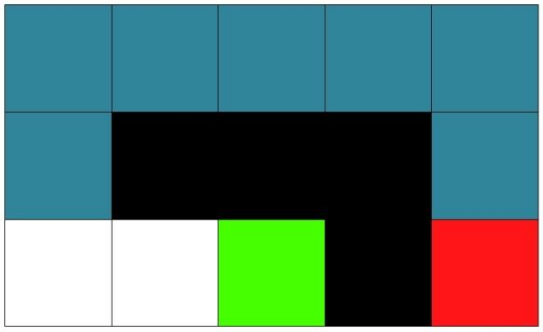
Untuk menampilkan hasil pencarian pada algoritma A* maka yang dilakukan adalah

1. Menemukan *parent* dari simpul akhir.



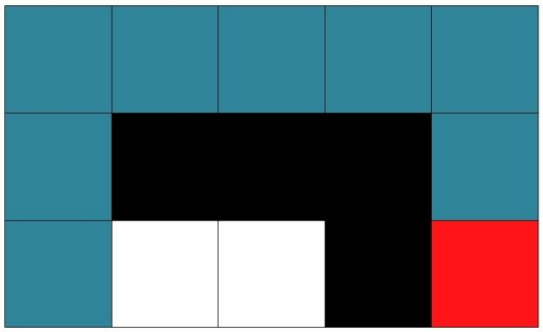
Gambar 3.19 – Awal menampilkan algoritma A*

2. Tidak menampilkan *parent* yang bukan simpul tetangga dari parent sebelumnya.



Gambar 3.20 – Menunggu simpul yang berdekatan untuk ditampilkan

3. Ulangi langkah ke 1 sampai menemukan simpul awal.

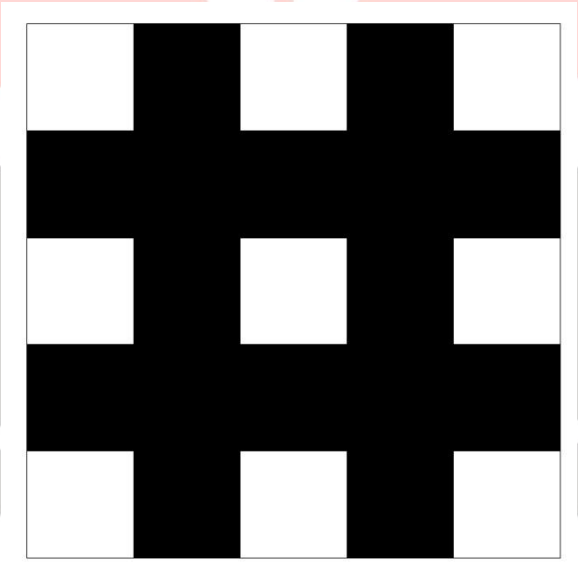


Gambar 3.21 – Hasil pencarian algoritma A* untuk simpul yang ditampilkan

3.2.3. Membuat *Block Maze*

Dalam membuat *block maze* sebagai permasalahan yang dipecahkan oleh algoritma A* dan *Breath First Search* maka dibuat suatu algoritma dalam membuat *block maze* tersebut secara acak. Hal-hal yang dilakukan didalam membuat *block maze* tersebut adalah

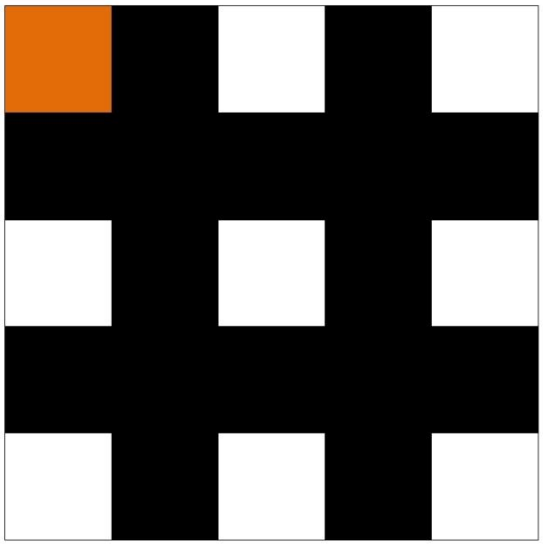
1. Tentukan seberapa banyak kotak yang diinginkan berdasarkan panjang dan lebarnya.
2. Buat jalur untuk simpul yang memiliki koordinat x atau y bernilai ganjil.



Gambar 3.22 – Menandai semua *path* yang ganjil pada pembuatan *Block Maze*

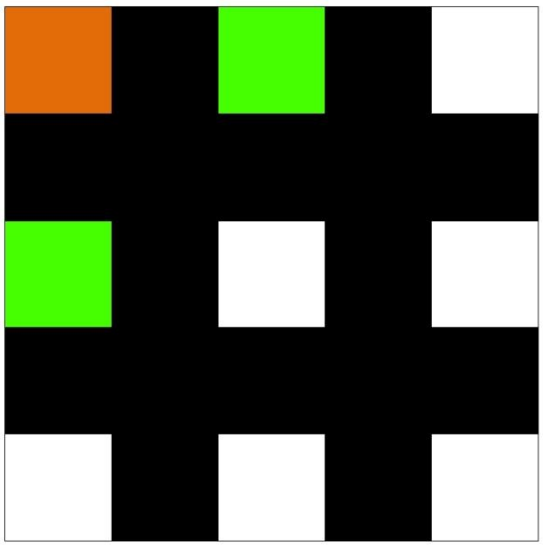
3. Masukkan simpul awal peta yang bernilai ganjil tersebut ke dalam tipe data *stack.*
4. Pilih secara acak simpul yang terdapat pada *stack* tersebut.

5. Jadikan status dari simpul yang terpilih tersebut sebagai simpul yang sudah terpilih.



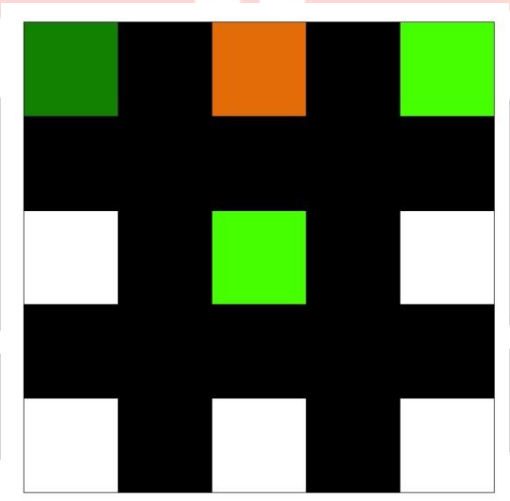
Gambar 3.23 - Awal memilih simpul pada pembuatan *Block Maze*

6. Hapus simpul yang terpilih tersebut dari *stack.*
7. Buka 4 arah setiap simpul tetangga dari simpul yang terpilih tersebut.



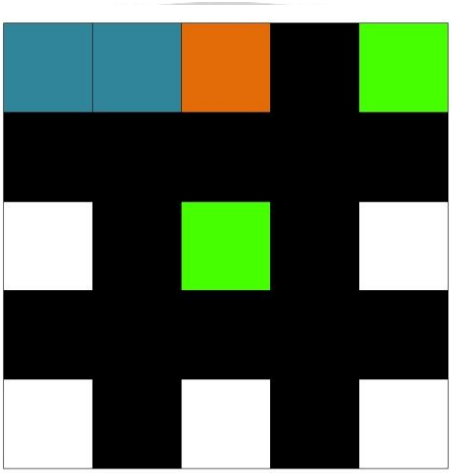
Gambar 3.24 – Membuka simpul-simpul tetangga yang terpilih pada pembuatan *Block Maze*

8. Cek masing-masing status dari setiap simpul-simpul tetangga tersebut:
   a. Apabila status dari simpul tersebut belum dikunjungi maka masukkan simpul tersebut ke *stack.*
   b. Apabila simpul tersebut berstatus telah dikunjungi maka masukkan ke daftar kandidat untuk melakukan penyambungan dari simpul sebelumnya.
9. Pada simpul yang terdapat pada daftar kandidat penyambungan maka akan:
   a. Jika daftar kandidat untuk melakukan penyambungan tidak kosong maka memilih secara acak dari simpul-simpul pada daftar kandidat tersebut dan lakukan penyambungan dengan simpul tetangganya.



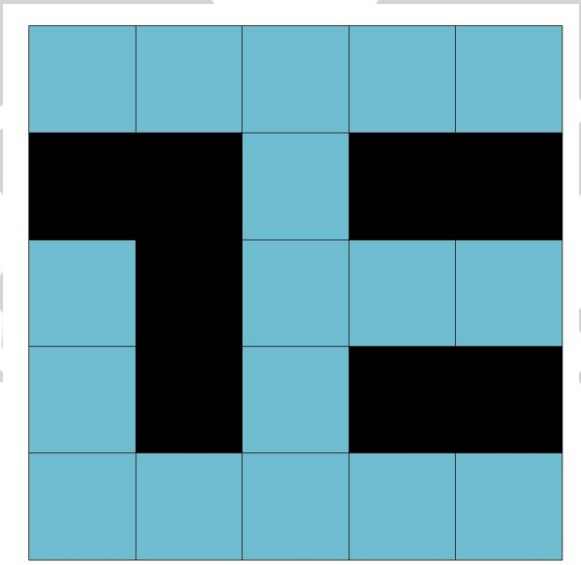
Gambar 3.25 – Simpul tetangga yang terpilih karena telah sebelumnya telah dikunjungi pada pembuatan *Block Maze*

b. Simpul yang telah tersambung statusnya akan diubah menjadi telah dikunjungi.



Gambar 3.26 – Simpul yang terpilih telah terhubung secara acak

10. Lakukan langkah ke 4 sampai semua simpul yang terdapat pada *stack* telah habis.



Gambar 3.27 – Hasil pembuatan *Block Maze*

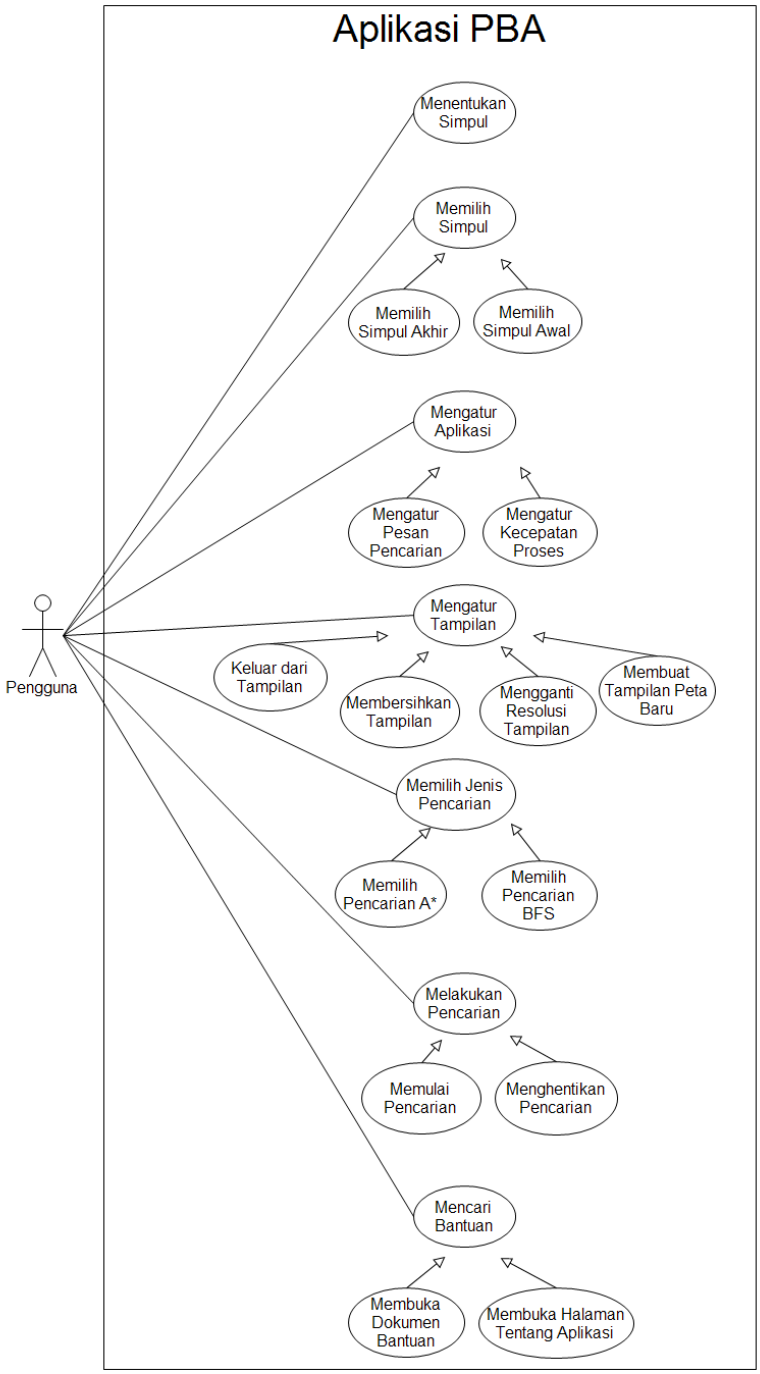
3.3 Analisis Proses

3.3.1 Model Sistem

Pemodelan pada perangkat lunak digambarkan kedalam bentuk UML. Pembangunan pada perangkat lunak ini, dibangun berdasarkan metodologi pengembangan perangkat lunak berbasis objek.

3.3.1.1 *Use Case Diagram*

Berikut ini adalah *use case* diagram yang digunakan pada aplikasi PBA.



Gambar 3.28 – *Use case diagram* PBA

Penjelasan mengenai *use case diagram* yang terdapat pada gambar 3.28 akan dijelaskan ke dalam tabel 3.1, yaitu

Tabel 3.1 - *Use case diagram* PBA

No	**Nama *Use Case***	***Flow of Events* untuk Individual *Use Cases***
1	Membuka Dokumen Bantuan	*Use case* ini adalah generalisasi abstrak, *use case* ini berhubungan dengan Membuka Dokumen Bantuan dan Membuka Halaman Tentang Aplikasi.
2	Menghentikan Pencarian	*Use case* ini terjadi saat pengguna menghentikan pencarian jalur.
3	Membersihkan Tampilan	*Use case* ini terjadi saat pengguna ingin menghilangkan hasil pencarian yang dilakukan oleh aplikasi.
4	Mengganti Resolusi Tampilan	*Use case ini* terjadi saat pengguna mengganti resolusi untuk *block maze.*
5	Membuka Dokumen Bantuan	*Use case* ini terjadi saat pengguna ingin membuka halaman bantuan untuk mempermudah penggunaan aplikasi.
6	Memilih Jenis Pencarian	*Use case* ini adalah generalisasi abstrak. *Use case* ini berhubungan dengan Memilih Pencarian A* dan Memilih Pencarian *Breadth first search*.
7	Mengatur Kecepatan Proses	*Use case* ini terjadi saat pengguna ingin mengatur waktu untuk menunggu aplikasi melakukan aksi selanjutnya.
8	Keluar dari Tampilan	*Use case* ini terjadi saat pengguna ingin keluar dari aplikasi.
9	Mengatur Tampilan	*Use case* mengatur tampilan adalah generalisasi abstrak. *Use case* ini berhubungan dengan pengaturan yang terdapat pada *block maze* yang terdiri dari Membersihkan Tampilan, Mengganti Resolusi Tampilan, Membuat Tampilan Peta Baru dan Keluar

No	Nama *Use Case*	*Flow of Events* untuk Individual *Use Cases*
		dari Tampilan.
10	Menentukan Simpul	*Use case* ini terjadi saat pengguna ingin menentukan simpul pada *block maze*.
11	Memulai Pencarian	*Use case* ini terjadi saat pengguna memulai melakukan pecarian jalur setelah pengguna menentukan simpul awal dan simpul akhir.
12	Memilih Simpul	*Use case* memilih simpul adalah generalisasi abstrak. *Use case* ini berhubungan dengan pemilihan tipe simpul yaitu Memilih Simpul Awal dan Memilih Simpul Akhir.
13	Melakukan Pencarian	*Use case* ini adalah generalisasi abstrak, *use case* ini berhubungan dengan Memulai Pencarian dan Menghentikan Pencarian.
14	Memilih Pencarian A*	*Use case* ini terjadi saat pengguna memutuskan untuk memilih pencarian A*
15	Memilih Pencarian *Breadth First Search*	*Use case* ini terjadi saat pengguna memutuskan untuk memilih pencarian *Breadth first search.*
16	Mengatur Aplikasi	*Use case* Mengatur Aplikasi adalah generalisasi abstrak. *Use case* ini berhubungan dengan Mengatur Pesan Pencarian dan Mengatur Kecepatan Proses.
17	Mengatur Pesan Pencarian	*Use Case* ini terjadi saat pengguna ingin melihat pesan pencarian yang dihasilkan oleh aplikasi PBA
18	Membuat Tampilan Peta Baru	*Use case* ini terjadi saat pengguna ingin membuat *block maze* baru.
19	Memilih Simpul Akhir	*Use case* ini terjadi saat pengguna menentukan simpul akhir pada *block maze.*
20	Memilih Simpul Awal	*Use case* ini dilakukan ketika pengguna ingin menentukan simpul awal pada peta *block maze.*
21	Membuka Halaman Tentang Aplikasi	*Use case* ini terjadi saat pengguna ingin mengetahui tentang informasi singkat mengenai aplikasi ini.

UNIVERSITAS PENDIDIKAN INDONESIA
PERPUSTAKAAN